안드로이드 정상 및 악성 앱 판별을 위한 최적합 머신러닝 기법

이형우[1*], 이한성[2]
[1]한신대학교 컴퓨터공학부 교수, [2]한신대학교 컴퓨터공학과 대학원생

Optimal Machine Learning Model for Detecting Normal and Malicious Android Apps

Hyung-Woo Lee[1*], HanSeong Lee[2]
[1]Professor, Div. of Computer Engineering, Hanshin University
[2]Graduate Student, Dept. of Computer Engineering, Hanshin University

요 약 안드로이드 플랫폼 기반 모바일 애플리케이션은 디컴파일이 간단하여 정상 앱과 유사한 악성 애플리케이션을 만들 수 있으며, 제작된 악성 앱은 안드로이드 서드 파티(third party) 앱 스토어를 통해 배포되고 있다. 이 경우 악성 애플리케이션은 기기 내 개인정보 유출, 프리미엄 SMS 전송, 위치정보와 통화 기록 유출 등의 문제를 유발한다. 따라서 최근 이슈가 되고 있는 머신러닝 기법 중에서 최적의 성능을 제공하는 모델을 선별하여 악성 안드로이드 앱을 자동으로 판별할 수 있는 기법을 제공할 필요가 있다. 이에 본 논문에서는 공인 실험 데이터셋을 이용하여 안드로이드 앱의 특징 정보를 선별한 후에 총 네 가지의 성능 평가 실험을 통해 안드로이드 악성 앱 판별에 최적의 성능을 제공하는 머신러닝 모델을 제시하였다.

주제어 : 모바일 단말, 안드로이드, 악성 애플리케이션, 특징 공학, 머신러닝, 랜덤 포레스트

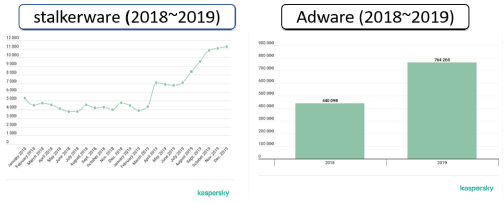
Abstract The mobile application based on the Android platform is simple to decompile, making it possible to create malicious applications similar to normal ones, and can easily distribute the created malicious apps through the Android third party app store. In this case, the Android malicious application in the smartphone causes several problems such as leakage of personal information in the device, transmission of premium SMS, and leakage of location information and call records. Therefore, it is necessary to select a optimal model that provides the best performance among the machine learning techniques that have published recently, and provide a technique to automatically identify malicious Android apps. Therefore, in this paper, after adopting the feature engineering to Android apps on official test set, a total of four performance evaluation experiments were conducted to select the machine learning model that provides the optimal performance for Android malicious app detection.

Key Words : Smart phones, Android, Malware, Feature Engineering, Machine Learning, Random Forest

This work was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) (NRF-2017R1D1B03035040)
*교신저자 : 이형우(hwlee@hs.ac.kr)
접수일 2020년 4월 20일 수정일 2020년 6월 10일 심사완료일 2020년 6월 23일

1. 서론

스마트폰이 대중화됨에 따라 지속적으로 급증하고 있는 안드로이드 악성 애플리케이션을 통해 사용자도 모르게 기기내 개인정보 유출, 프리미엄 SMS 전송, 위치정보와 통화 기록 유출 등의 문제가 발생하고 있다. Symantec에서 발표한 “2018 Internet Security Threat Report”에 따르면 2016년부터 2017년 사이 모바일 악성 앱 변종이 54% 증가하였으며, 일평균 24,000여개의 악성 앱이 발견되었다[1]. 또한 Kaspersky에서 조사한 “Mobile malware evolution 2019”에 따르면 해당 모바일 제품에서 총 3,503,952개 악성 형태의 설치 패키지가 탐지되었고, 69,777개의 신규 모바일 뱅킹 트로이목마와 68,362개의 모바일 랜섬웨어가 발견되었다고 밝혔다[2]. 이처럼 안드로이드 모바일 단만을 대상으로 신규 악성 앱이 지속적으로 출현함과 동시에 변종 악성 애플리케이션 역시 급증하고 있는 추세이다. 이는 안드로이드 플랫폼 기반 모바일 애플리케이션이 디컴파일이 쉬운 Java 언어로 개발되어 누구라도 손쉽게 정상 앱과 유사한 악성 애플리케이션을 만들 수 있기 때문이며, 제작된 악성 앱을 안드로이드 서드 파티(third party) 앱 스토어를 통해 손쉽게 배포할 수 있기 때문이다. 따라서 본 논문에서는 기존 기법의 문제점 분석 결과를 토대로, 안드로이드 악성 앱 판별에 가장 적합한 머신러닝 기법 등을 도출하여 정상 앱과 변종 악성 앱을 자동으로 판별하는 기법을 도출하고자 한다.



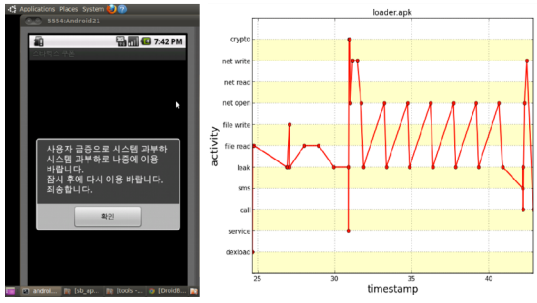
[Fig. 1] Number of Detected Stalkerware and Adware in 2018 and 2019 (Kaspersky)

2. 관련 연구

2.1 기존의 안드로이드 앱 분석 기법

현재까지 안드로이드 악성 앱을 분석하는 방법은 크게 정적 분석과 동적 분석 기법으로 나눌 수 있다. 안드로이드 애플리케이션 내에 포함된 DEX 파일은 디스어셈블리 또는 디컴파일하면 내부 코드를 획득할 수 있기 때문에 코드를 중심으로 행위를 정적 분석할 수 있다. 또한 모바일 단말 또는 애뮬레이터 내에 설치된 안드로이드 애플리케이션을 실행하여 발생하는 시스템 호출(system call) 등을 모니터링하여 동적 분석할 수 있다. 동적 분석 방법은 DroidBox 등의 도구를 사용하며 난독화 또는 암호화가 된 앱에 대한 분석도 가능하다는 장점이 있고, 정적 분석은 Androguard 등과 같은 도구를 사용하여 APK 파일을 디컴파일하고 수집된 앱 내부 정보를 토대로 분석 과정을 수행하기 때문에 동적 분석 방식 보다 상대적으로 다양한 특징 정보 수집이 가능하다.

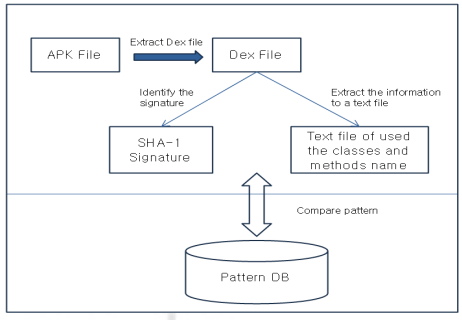
따라서, 머신러닝 기법과 연계하기 위해서는 정적 분석 기법을 이용하여 안드로이드 애플리케이션에 포함된 AndroidManifest.xml 파일과 classes.dex 파일로부터 속성 정보를 추출하고 정상과 악성 애플리케이션의 특징을 선별하는 과정을 수행할 수 있다.



[Fig. 2] Dynamic Analysis of Android Applications Using the Droidbox Tool

2.2 기존의 머신러닝 기반 악성 앱 탐지 연구

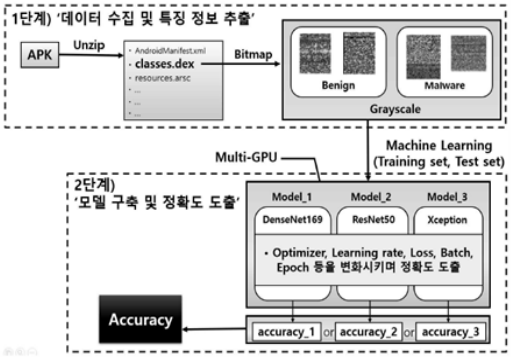
악성 앱 탐지를 위한 기존 연구로는 APK 또는 DEX 파일의 해시값을 이용한 시그니처(signature) 기반의 탐지가 가장 대표적이다. 하지만, 이는 리패키징을 이용하여 탐지를 우회할 수 있다는 단점이 존재한다.



[Fig. 3] Detection Classes and Methods Through Mapping Process in Pattern DB[3]

따라서 시그니처 기반 탐지를 보완하기 위해 DEX 파일 내 클래스와 메소드 이름을 추출하여 DB 내 매핑 과정을 통한 탐지 기법[3]이 제시되었으나, 클래스와 메소드를 비교하는 과정에서 오버헤드가 발생한다는 단점이 발생한다.

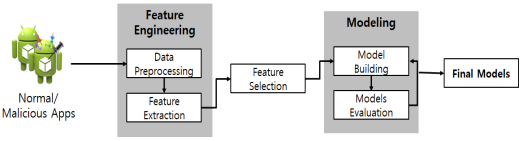
앞서 접근한 방식과는 다른 악성 앱 탐지 기법으로 DEX 파일을 비트맵화하여 머신러닝 기반의 안드로이드 악성 앱 탐지 기법[4]이 제시되었다. CNN 알고리즘을 적용하여 최대 84.73% ~ 87.96%의 정확도로 검출하였으나, 이 기법인 경우 머신러닝 학습 과정에 필요한 정상과 악성 앱 특징 정보를 자동 선별하기가 모호하다는 단점이 있으므로 이에 대한 개선 과정이 제시되어야 한다.



[Fig. 4] Bitmap of Dex Files to Detect Malicious Apps Based on Machine Learning[4]

3. 악성 앱 판별을 위한 특징 공학 기반의 머신러닝

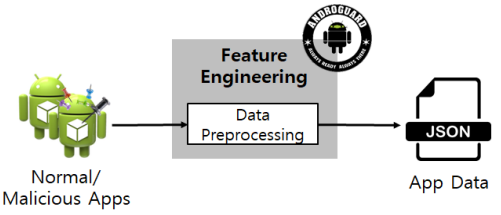
정상 또는 악성 여부를 알 수 없는 안드로이드 애플리케이션이 입력으로 들어왔을 때 특징 공학(feature engineering)을 적용하여 애플리케이션의 속성과 특징을 추출하고, 안드로이드 정상/악성 앱 판별에 최적의 머신러닝 모델을 선택할 필요가 있다. 따라서 최적의 머신러닝 기법을 적용한다면 정상과 악성 앱에 대한 판별의 정확도를 향상시킬 수 있을 것으로 예상된다. 따라서 안드로이드 악성 앱을 판별하기에 가장 적합한 머신러닝 모델을 구축하기 위해서는 아래 [Fig. 5]와 같이 정적 분석 기법을 기반으로 안드로이드 애플리케이션의 특성을 도출하는 전처리 과정과 특징 공학 과정을 수행하고, 최적의 성능을 제시하는 머신러닝 모델 선별하는 과정이 필요하다.



[Fig. 5] Machine Learning Process

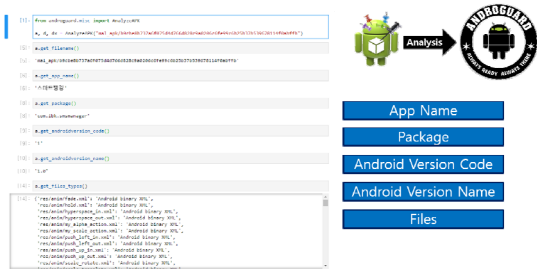
3.1 안드로이드 앱 내부 데이터 전처리

APK 파일은 안드로이드 응용 프로그램 패키지로 ZIP 파일 기반으로 데이터가 압축되어 있다. 그러므로 APK 파일에 있는 데이터를 추출하기 위해서는 압축해제 후 각 파일에서 데이터를 추출해야 한다. 안드로이드 APK 파일에 대해 데이터를 전처리하기 위해 Androguard 파이썬 라이브러리[5]를 사용하였다. Androguard는 안드로이드 기반의 애플리케이션을 분석할 수 있는 역공학 도구이다. 파이썬 기반의 모듈로 DEX, ODEX, APK, 안드로이드 바이너리 분석, 안드로이드 리소스, DEX/ODEX 바이트 코드 디어셈블리, DEX/ODEX 파일 디컴파일 기능을 제공한다. Androguard를 이용하여 APK에 존재하는 패키지명, 퍼미션 정보, 파일 목록 등을 추출하여 JSON 파일로 저장하기 위해 다음 [Fig. 6]과 같이 데이터 전처리 과정을 수행하였다. 전처리 과정에서는 Androguard 라이브러리를 이용하여 정적 분석 과정을 수행하며 APK 내부에 포함된 특징 정보를 추출하도록 구현하였다.

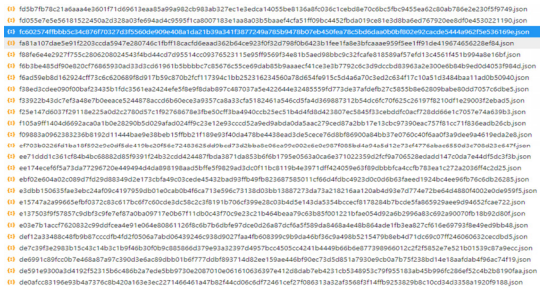


[Fig. 6] Data Preprocessing for Feature Engineering

다음 [Fig. 7]은 Androguard 라이브러리를 이용해 Fakebank 앱을 대상으로 앱의 이름과 패키지명, 버전 코드, 버전 이름, 파일 목록 등을 추출한 결과이다. Androguard 라이브러리의 AnalyzeAPK 클래스로 앱에서 데이터를 추출하는 함수를 [Fig. 8]과 같이 구현하였다.



[Fig. 7] Extracting Properties from 'Fakebank' App Using Androguard



[Fig. 9] JSON Files Created by Preprocessing 6,000 APK Files

```
def extract_attributes(self, fileName):
    a, d, dx = AnalyzeAPK(fileName)

    attributes = {}

    attributes['sha256'] = self.get_hash_digest(sha256, a.get_raw())
    attributes['sha512'] = self.get_hash_digest(sha512, a.get_raw())
    attributes['appName'] = a.get_app_name()
    attributes['package'] = a.get_package()
    attributes['androidVersionCode'] = a.get_androidversion_code()
    attributes['androidVersionName'] = a.get_androidversion_name()
    attributes['files'] = [key for key, value in a.get_files_types().items()]
    attributes['dexNames'] = [name for name in a.get_dex_names()]
    attributes['isMultiDex'] = a.is_multidex()
    attributes['mainActivity'] = a.get_main_activity()
    attributes['mainActivities'] = list(a.get_main_activities())
    attributes['activities'] = a.get_activities()
    attributes['services'] = a.get_services()
    attributes['receivers'] = a.get_receivers()
    attributes['providers'] = a.get_providers()
    attributes['permissions'] = a.get_permissions()
    attributes['maxSDKVersion'] = a.get_max_sdk_version()
    attributes['minSDKVersion'] = a.get_min_sdk_version()
    attributes['targetSDKVersion'] = a.get_target_sdk_version()
    attributes['libraries'] = a.get_libraries()
    attributes['androidManifest'] = a.get_android_manifest_axml().get_xml().decode()
    attributes['isAsciiObfuscation'] = analysis.is_ascii_obfuscation(d[0]) if len(d) > 0 else False

    attributes['classes'] = list(dx.classes.keys())
    attributes['methods'] = [str(m.method) for m in dx.methods.values()]
    attributes['strings'] = list(dx.strings.keys())

    for key, value in attributes.items():
        if value is None:
            attributes[key] = ''

    return attributes
```

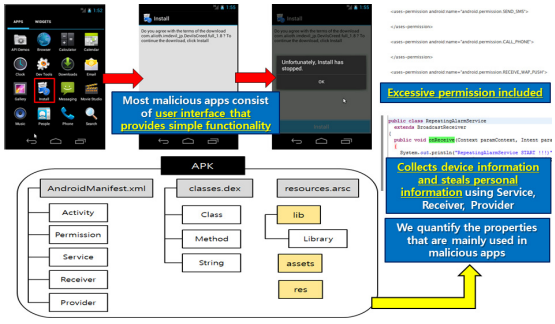
[Fig. 8] Implementing a Function to Extract the Property Values of the App Using Androguard

[Fig. 10] Content of JSON File of 'Talk to Me Cloud' App

안드로이드 앱 APK 파일을 구현한 함수를 이용하여 모두 JSON 파일로 저장하였다. JSON 파일의 이름은 [Fig. 9]과 같이 분석한 APK 파일의 SHA512 값으로 생성토록 설정하였다. Talk To Me Cloud 앱의 JSON 파일을 열면 다음 [Fig. 10]과 같이 앱의 속성값을 key-value 형태로 확인할 수 있다. 그리고 정상과 악성 여부를 malware 속성으로 추가(malware 필드 속성 값이 false이면 정상 앱을 의미함)하여 데이터셋에 대한 판별/분류 기준을 속성 정보로 포함시켰다.

3.2 안드로이드 앱 특징 정보 선별

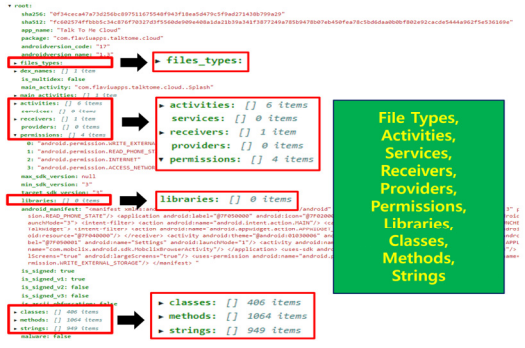
특징 추출 및 선별 과정을 하기 위해서는 안드로이드 내부 구조 및 안드로이드 악성 애플리케이션이 갖는 특징에 대한 이해가 필요하다. 대부분의 악성 애플리케이션은 [Fig. 11]과 같이 단순 기능을 제공하는 사용자 인터페이스로 구성되어 있으면서도 정상 애플리케이션과는 달리 과도한 퍼미션을 포함한 경우가 많다. 특히 악성 APK인 경우 Service, Receiver, Provider 퍼미션을 이용해 기기의 정보 수집 및 개인정보 탈취 행위를 수행한다. 이러한 안드로이드 악성 애플리케이션의 특징을 토대로 안드로이드 APK에서 추출 가능한 특성 정보에 대한 최종적인 선별 과정을 수행하였다. AndroManifest.xml 파일에서 Activity, Permission, Service, Receiver, Provider 속성을 추출한 후 classes.dex 파일에서 Class, Method, String 정보를 추가하였으며, lib 폴더에 있는 Library 정보와 기타 APK 파일 내에 있는 파일 목록 등을 특성 정보로 구성하였다.



[Fig. 11] Feature Selection and Extraction Process

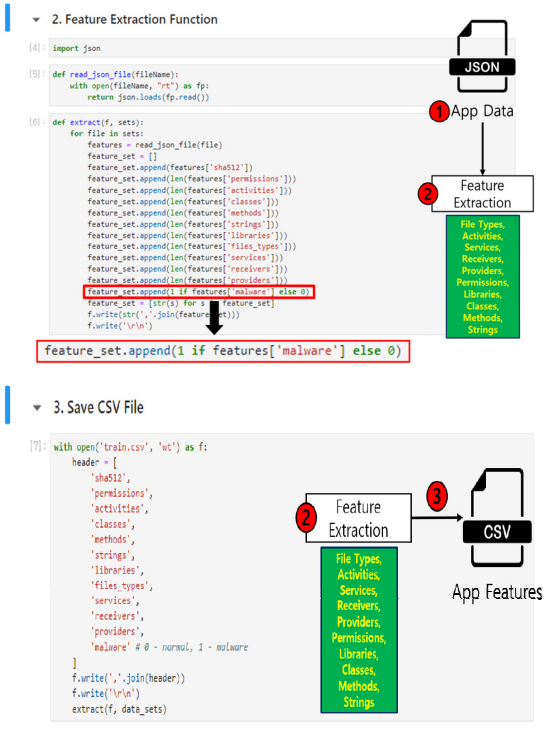
3.3 안드로이드 앱 특징 정보 수치화

APK 파일내 전체 특징 정보를 포함하고 있는 JSON 파일로부터 정상/악성 앱 판별에 사용될 수 있는 특징 정보를 선별한 후에 이를 다시 수치화하는 과정을 수행하였다. 추출된 APK의 속성 중 APK 안에 존재하는 파일(File Types), 액티비티(Activities), 서비스(Services), 리시버(Receivers), 프로바이더(Providers), 라이브러리(Libraries), DEX 클래스(Classes), DEX 메소드(Methods) 및 DEX 문자열(Strings)의 개수를 특징 정보로 각각 수치화하는 과정을 구현하였다.



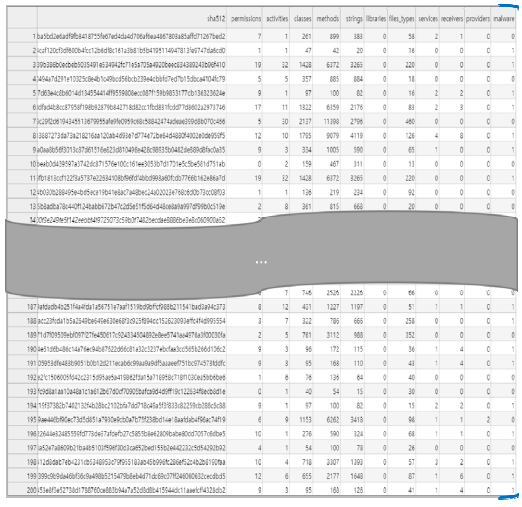
[Fig. 12] Selected Features

앞서 제시한 바와 같이 안드로이드 APK로부터 추출된 전처리된 정적 특징 정보를 대상으로 선별 과정을 거친 후 최종적으로 수치값으로 변환하는 함수를 [Fig. 13]과 같이 구현하였다. 또한 해당 앱을 식별할 수 있도록 SHA512 기반 해시값을 포함했으며, 머신러닝 학습 결과 정상/악성 판별 정보를 포함할 수 있도록 필드를 추가하였다.



[Fig. 13] Implementation of a Function to Extract Features from Pre-processed Data and Saving Features Extracted from JSON Files as CSV Files

200개의 앱에서 10개의 특징을 추출하여 이를 수치화한 후 머신러닝 학습 데이터(train.csv) 파일로 저장한 결과는 다음 [Fig. 14]와 같다.



[Fig. 14] Extracting 10 Features from Each of 200 Apps and Saving It as a CSV File

4. 머신러닝 최적 모델 선정

4.1 실험 환경 및 실험 개요

안드로이드 악성 앱 판별을 위한 머신러닝 실험 환경은 다음 〈Table 1〉과 같다. 운영체제는 Ubuntu Server를 기반으로 하였으며 시스템 내부에 Jupyter Notebook과 Jupyter Lab을 설치하여 웹 환경에서 파이썬 프로그래밍을 할 수 있는 통합개발환경, 그리고 파이썬 기반으로 안드로이드 APK 파일을 분석할 수 있는 Androguard 라이브러리와 머신러닝 모델을 구현하기 위한 scikit-learn 라이브러리를 설치하여 각각의 머신러닝 모델별로 안드로이드 정상/악성 앱 판별 성능을 비교 분석하였다.

〈Table 1〉 Machine Learning Environment

Environment	Description
CPU	Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz
RAM	32GB
OS	Ubuntu Server 18.04.3 LTS
Jupyter Notebook 6.0.1[6]	Web-based interactive computing open source software
Jupyter Lab 1.2.6[7]	Web-based user interface with improved Jupyter
Python 3.6[8]	interactive programming language
Androgaurd 3.4.0[5]	Android APK file analysis library
scikit-learn 0.22[9]	Python-based machine learning library

안드로이드 정상/악성 앱 판별에 적합한 최적의 머신러닝 모델을 선정하는 과정이 필요하다. 실험의 목적은 안드로이드 APK로부터 추출 및 수치화된 특징 정보를 대상으로 최고의 판별 성능을 보이는 머신러닝 모델을 찾는 것이다. 이를 위해 첫 번째로 실험 대상이 되는 안드로이드 APK 대상을 늘려 나갔을 경우 판별 성능을 비교하였고, 두 번째로는 악성 분류에 사용 가능한 퍼미션의 특징 정보의 개수를 달리하면서 각각의 머신러닝 모델별로 정확도(accuracy)와 F1 스코어를 비교하여 안드로이드 정상/악성 앱 판별에 적합한 최적의 머신러닝 모델을 선정하는 과정을 수행하였다.

안드로이드 정상/악성 앱 판별에 사용 가능한 머신러닝 모델을 선정하기 위해 〈Table 2〉와 같이 총 8개의 머신러닝 모델과 옵션을 대상으로 각각의 모듈을 이용하였을 경우 판별 성능을 비교 분석하였다.

〈Table 2〉 List of Machine Learning Models Used

No.	Model	Options
1	SVM	Random state=0
2	Decision Tree	Random state=0
3	Ada Boost	Random state=0
4	Random Forest	Random state=0
5	Gradient Boosting	Random state=0
6	Gaussian Naïve Bayes	-
7	Logistic Regression	Random state=0
8	k-NN	N neighbors=2

4.2 머신러닝 성능 평가 지표

머신러닝 모듈을 적용하였을 경우 안드로이드 정상/악성 앱에 대해 〈Table 3〉과 같이 네 가지 형태의 혼동행렬(Confusion matrix) 분류 결과가 나타나게 된다. 따라서 각각의 혼동 행렬의 결과값을 토대로 사용된 각각의 머신러닝 모델에 대한 판별 성능을 비교 평가할 수 있으며, 이를 토대로 안드로이드 APK 판별에 가장 적합한 머신러닝 모델을 선별할 수 있는 정량적 성능평가 지표로 사용하였다.

〈Table 3〉 Confusion Matrix

	Predicted : Yes	Predicted : No
Malicious APK	True Positive	False Negative
Normal APK	False Positive	True Negative

일반적으로 머신러닝 모델의 성능을 비교/평가하는 지표로는 정확도(Accuracy), 정밀도(Precision), 재현율(Recall), F1 score(F-Score or F-measure) 등이 있다. 이 중에서 F1 score는 정밀도와 재현율을 모두 반영하는 평가지표로 머신러닝 모델의 성능 평가지표로 가장 많이 사용하고 있다.

$$Accuracy = \frac{(TP + TN)}{100} \quad (1)$$

$$\mathrm{Precision} = \frac{TP}{(TP + FP)} \quad (2)$$

$$Recall = \frac{TP}{(TP + FN)} \quad (3)$$

$$F1\,score = \frac{2\,(\mathrm{Precision} * Recall)}{(\mathrm{Precision} + Recall)} \quad (4)$$

따라서, 안드로이드 정상/악성 앱 판별 성능을 비교하기 위해 〈Table 3〉에 제시된 8가지 머신러닝 모델을 대상으로 각 머신러닝 모델별로 안드로이드 정상/악성앱 판별 정확도와 F1 스코어를 측정하여 최적의 머신러닝 모델을 도출/선별하는 과정을 수행하였다.

5. 머신러닝 기반 안드로이드 악성 앱 판별 성능 비교 평가

5.1 머신러닝 성능 평가 실험 데이터

실험 데이터로는 2018 정보보호 R&D 데이터 챌린지에서 오픈한 데이터셋[10,11]을 사용하였다. 인터넷에서 다운 가능한 KU-CISC2018-Android-Pre-Train 파일 내에는 정상 앱 4,000개와 악성 앱 2,000개 등 총 6,000개의 APK 파일이 포함되어 있다. 이에 〈Table 4〉와 같이 사용된 특징과 데이터 수를 각기 달리 하며 총 네 가지 방법의 실험을 수행하였다.

〈Table 4〉 Number of APK Data and Features Used per Experiments

Experiment No.	Number of APK	Number of Features
1	200	9
2	5,992	9
3	5,992	13
4	4,700	429

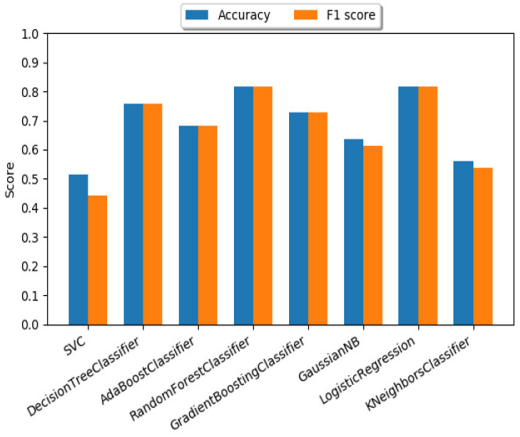
5.2 실험 1 (200개 데이터, 9개 특징 사용)

실험 1은 2018 정보보호 R&D 데이터 챌린지에서 오픈한 데이터셋 6,000개 중에서 200개의 안드로이드 APK 파일을 대상으로 머신러닝 기반 판별 성능을 비교하였다. 학습용 APK 134개, 테스트용 APK 66개를 각각 사용하였고, 〈Table 5〉와 같이 각각의 안드로이드 APK로부터 각각 9개의 특징을 추출하여 머신러닝 학습 및 판별 과정에 사용하였다.

〈Table 5〉 Used Features on Experiment #1

No.	Feature Name
1	Number of Permissions
2	Number of Activities
3	Number of Classes
4	Number of Strings
5	Number of Librarires
6	Number of Files
7	Number of Services
8	Number of Receivers
9	Number of Providers

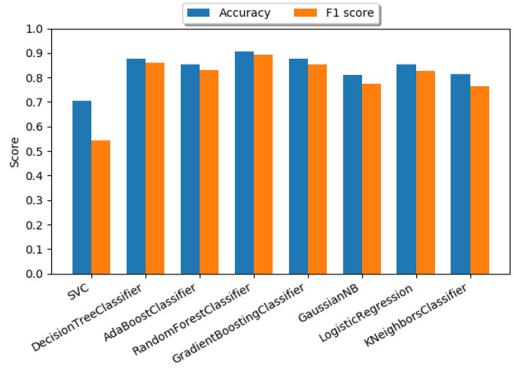
총 200개의 APK 파일을 대상으로 실험한 결과 [Fig. 15]와 같이 랜덤 포레스트와 로지스틱 회귀 머신러닝 모델이 대체로 높은 정확도(81.82%)를 보였으며, F1 스코어는 랜덤 포레스트가 로지스틱 회귀 모델(81.75%)보다 상대적으로 조금 더 높은 판별 성능을 나타냈다. 하지만 대부분의 머신러닝 모델에서 대체로 낮은 판별 성능을 나타냈는데, 이는 적은 개수의 APK 파일을 대상으로 머신러닝 학습/판별 과정을 수행하였기 때문에 결과적으로 판별 지표가 상대적으로 낮다는 것을 알 수 있었다.



[Fig. 15] Discriminating Performance Comparison of ML Models on Experiment #1

5.3 실험 2 (5,992개 데이터, 9개 특징 사용)

실험 2는 총 5,992개의 APK 파일(학습 데이터 4,014개, 테스트 데이터 1,978개를 각각 사용)을 대상으로 8개의 머신러닝 모델별 안드로이드 악성 앱 판별 성능을 비교/분석하였다. 실험 2에서는 실험 1과 동일하게 9개의 특징 정보를 사용하여 APK 파일 수를 증가하였을 경우 정확도 및 F1 스코어 값을 측정하였다. 실험 결과 [Fig. 16]와 같이 랜덤 포레스트(90.68%)가 가장 높은 판별 성능을 보였으며, F1 스코어 역시 랜덤 포레스트 머신러닝 모델이 89.21%로 실험 1 보다 약 7% 정도 판별 성능이 향상된 것을 알 수 있다.



[Fig. 16] Discriminating Performance Comparison of ML Models on Experiment #2

5.4 실험 3 (5,992개 데이터, 13개 특징 사용)

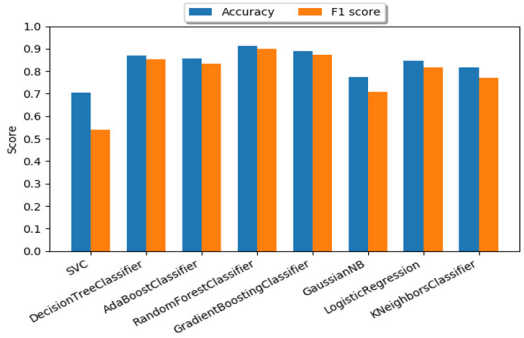
실험 3은 실험 2와 동일하게 총 5,992개의 데이터를 사용하였으며 〈Table 6〉과 같이 총 13개의 프로텍션 레벨 퍼미션을 특징 정보로 사용하였다. 프로텍션 레벨은 퍼미션에 잠재적 위험을 특성화한 속성으로 normal, signature, dangerous, signatureOrSystem 값을 갖는 특징 정보로 최근 안드로이드 앱에서 상대적으로 많이 사용하고 있는 퍼미션에 해당한다[12].

Androguard 라이브러리에서 제공하는 기능을 활용하여 각 APK 파일에 포함된 전체 퍼미션 정보를 대상으로 프로텍션 레벨 정보를 추출하고 각 퍼미션별 전체 개수를 수치화하는 함수 추가로 구현하였다. 이를 토대로 총 5,992개의 APK 파일내 포함된 프로텍션 레벨 퍼미션 정보를 토대로 8개의 머신러닝 모델별 악성/정상 앱 판별 성능을 비교/분석하였다.

〈Table 6〉 Used Features on Experiment #3

No.	Feature Name
1	Number of Permissions
2	Number of Normal Permissions
3	Number of Dangerous Permissions
4	Number of Signature Permissions
5	Number of Signature Or System Permissions
6	Number of Activities
7	Number of Classes
8	Number of Strings
9	Number of Libraries
10	Number of Files
11	Number of Services
12	Number of Receivers
13	Number of Providers

실험 결과인 [Fig. 17]과 같이 랜덤 포레스트가 다른 머신러닝 모델이 비해 상대적으로 높은 판별 성능(정확도 91.35%)을 나타냈으며, F1 스코어는 89.99%로 실험 2와 비교하여 대략 0.7% 증가한 성능을 보였다. 따라서 실험 결과 프로텍션 레벨을 특징으로 사용하더라도 안드로이드 정상/악성 앱에 대한 판별 성능에는 큰 영향을 미치지 않는다는 것을 확인할 수 있었다.



[Fig. 17] Discriminating Performance Comparison of ML Models on Experiment #3

5.5 실험 4 (4,700개 데이터, 429개 특징 사용)

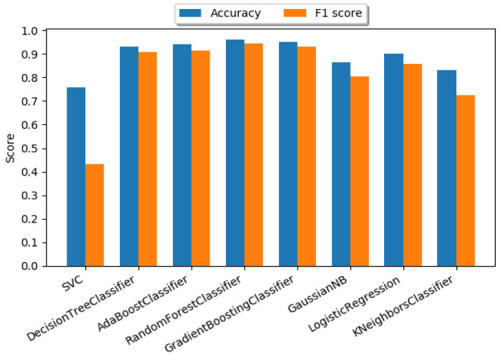
실험 4에서는 4,700개의 APK 파일을 사용하였으며 전체 실험 파일 중 66.7%인 3,149개는 학습 데이터로, 33.3%인 1,551개는 테스트 데이터로 사용하였다. 실험 3(5,992개 데이터 사용)보다 실험 파일의 수가 4,700개로 데이터 수가 줄어든 이유는 일부 앱에서 퍼미션 이름이 잘못 기재된 앱을 실험 대상에서 제외하였기 때문이다. 그리고 앞서 제시한 실험 3과는 달리 안드로이드에서 사용 가능한 퍼미션 421개를 모두 특징으로 사용(〈Table 7〉)하였다. 이를 위해 421개의 퍼미션 중 사용중인 퍼미션은 1로, 사용하지 않는 퍼미션은 0으로 인코

〈Table 7〉 Used Features on Experiment #4

No.	Feature Name
1	android.intent.category.MASTER_CLEAR.permission.C2D_MESSAGE
2	android.permission.ACCESS_ALL_EXTERNAL_STORAGE
	...
421	com.android.voicemail.permission.WRITE_VOICEMAIL
422	Number of Activities
423	Number of Classes
424	Number of Strings
425	Number of Librarires
426	Number of Files
427	Number of Services
428	Number of Receivers
429	Number of Providers.

딩하는 함수를 scikit-learn의 LabelEnconder를 활용하여 추가로 구현하였다.

실험 결과 [Fig. 18]과 같이 랜덤 포레스트가 가장 좋은 정확도 성능(96.13%)을 보였고, F1 스코어는 94.57%로 실험 3과 비교하면 약 4.5% 증가한 것을 확인할 수 있었다. 따라서, 실험 3 보다 실험 대상 APK 파일의 수가 줄었음에도 프로텍션 레벨 보다는 각각의 APK에 포함된 모든 퍼미션 정보를 특징 정보로 사용했을 경우 상대적으로 더 높은 판별 성능을 제공함을 확인할 수 있었다.



[Fig. 18] Discriminating Performance Comparison of ML Models on Experiment #4

또한, 실험 3과 비교해보면 SVM, k-NN 모델을 제외한 다른 6개의 머신러닝 모델에서 정확도 및 F1 스코어 등이 향상되었음을 확인할 수 있었다.

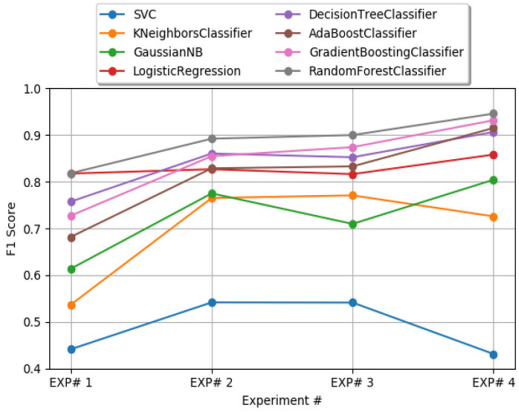
5.6 실험 결과 비교 분석

네 가지의 실험 결과 안드로이드 악성/정상 앱 판별 성능 지표 F1 스코어를 서로 비교하면 〈Table 8〉과 같다.

〈Table 8〉 Experimental Results

Classifier	EXP #1	EXP #2	EXP #3	EXP #4
SVM	44.13	54.16	54.13	43.12
Decision Tree	75.74	86.04	85.24	90.64
Ada Boost	68.17	82.87	83.28	91.50
Random Forest	81.82	89.21	89.98	94.57
Gradient Boosting	72.73	85.46	87.40	93.15
Gaussian Naïve Bayes	61.37	77.50	70.97	80.42
Logistic Regression	81.75	82.68	81.63	85.81
k-NN	53.67	76.52	81.80	72.59

8개의 머신러닝 모델을 대상으로 F1 스코어를 비교하였을 경우 [Fig. 19]와 같이 랜덤 포레스트가 네 번의 실험 모두에서 가장 좋은 성능을 보이는 것을 확인할 수 있었다. 또한 기존의 블록체인 기반 악성 앱 탐지 시스템 [13][14] 보다도 개선된 기능을 제공한다.



[Fig. 19] F1 Scores on ML Model based Detection of Normal/Malicious Android Apps

6. 결론

스마트폰 내 안드로이드 악성 애플리케이션을 통해 사용자도 모르게 기기 내 개인정보 유출, 프리미엄 SMS 전송, 위치정보와 통화 기록 유출 등의 문제가 발생하고 있다. 따라서 최근 이슈가 되고 있는 머신러닝 기법 중에서 최적의 성능을 제공하는 모델을 선별하여 악성 안드로이드 앱을 자동으로 판별할 수 있는 기법을 제공할 필요가 있다. 이에 본 논문에서는 공인 실험 데이터를 대상으로 총 네 가지 형태의 성능 평가 실험을 통해 가장 높은 성능을 나타내는 머신러닝 모델을 도출하였다. 실험 결과 랜덤 포레스트 모델을 이용할 경우 안드로이드 정상/악성 앱 판별 과정에 최적의 성능(94.57%)을 제공한다는 것을 확인할 수 있었다. 따라서 앞으로 랜덤 포레스트 머신러닝 모델을 이용하여 개인정보 유출 방지 및 안전성이 향상된 스마트폰 이용 환경을 제공할 수 있을 것으로 예상되며, 머신러닝 모델을 적용하여 자동 판별 결과를 [15]와 같은 블록체인 시스템 내에 저장/관리한다면 더욱더 보안성이 향상된 모바일 앱 이용 환경을 제공할 수 있을 것으로 기대된다.

REFERENCES

[1] Symantec. Internet Security Threat Report. Volume 23. March 2018. https://docs.broadcom.com/doc/istr-23-2018-en.

[2] Victor Chebyshev. Mobile malware evolution 2019. February 25, 2020. http://securelist.com/mobile-malware-evolution-2019/96280/.

[3] D.H.Park, E.J.Myeong and J.B.Yun, "Efficient Detection of Android Mutant Malwares Using the DEX file", Korea Institute Of Information Security And Cryptology, Vol.26, No.4, pp.895-902, 2016.

[4] D.H.Kim, M.G.Lee, M.S.Song and S.J.Cho, "Machine Learning based Android Malware Detection using Gray Scale Images", KOREA INFORMATION SCIENCE SOCIETY, Vol.45, No.1, pp.1245-1247, 2018.

[5] Androguard. https://github.com/androguard/androguard.

[6] Jupyter Notebook. https://jupyter.org/.

[7] Jupyter Lab. https://github.com/jupyterlab/jupyterlab.

[8] Python. https://www.python.org/.

[9] scikit-learn. https://scikit-learn.org/.

[10] J.W.Jang, J.S.Yun, A.Mohaisen, J.Y.Woo and H.K.Kim. "Detecting and classifying method based on similarity matching of Android malware behavior with profile.", SpringerPlus, Vol.5, No.1, pp.273, 2016.

[11] J.S.Yun, J.W.Jang, and H.K.Kim. "Andro-profiler: anti-malware system based on behavior profiling of mobile malware.", Journal of the Korea Institute of Information Security & Cryptology, Vol.24, No.1, pp.145-154, 2014.

[12] Android Documentation. http://developer.android.com/guide/topics/manifest/permission-element.html.

[13] S.M.Hwang and H.W.Lee, "Identification of Counterfeit Android Malware Apps using Hyperledger Fabric Blockchain," Journal of Internet Computing and Services, vol. 20, no. 2, pp. 61-68, 2019. DOI: 10.7472/jksii.2019.20.2.61.

[14] H.S.Lee and H.W.Lee, "Consortium Blockchain based Forgery Android APK Discrimination DApp using Hyperledger Composer," Journal of Internet Computing and Services, vol. 20, no. 5, pp. 9-18, 2019. DOI: 10.7472/jksii.2019.20.5.9.

[15] K.W.Bae, K.H.Lee, "Security of Database Based On Hybrid Blockchain," Journal of The Korea Internet of Things Society, Vol.6, No.1, pp.9-15, 2020. https://doi.org/10.20465/KIOTS.2020.6.1.009

이 형 우(Hyung-Woo Lee) [종신회원]

- 1994년 2월 : 고려대학교 컴퓨터학과 (학사)
- 1996년 2월 : 고려대학교 컴퓨터학과 (석사)
- 1999년 2월 : 고려대학교 컴퓨터학과 (박사)
- 1999년 3월 ~ 2003년 2월 : 백석대학교 정보통신학부 교수
- 2003년 3월 ~ 현재 : 한신대학교 컴퓨터공학부 교수

〈관심분야〉
사물인터넷, 정보보호, 모바일 보안 및 디지털 포렌식

이 한 성(HanSeong Lee) [준회원]

- 2017년 2월 : 한신대학교 컴퓨터공학부 졸업
- 2018년 9월 ~ 현재 : 한신대학교 일반대학원 컴퓨터공학과 석사과정

〈관심분야〉
블록체인 기술, 모바일 보안, 디지털포렌식, 리버스 엔지니어링 등